Variables and Binding

Mattox Beckman

beckman@iit.edu

Illinois Institute of Technology

Variables have many different attributes. These attributes can become *bound* to the variable at different times.

- Know the difference between static and dynamic binding...
  - of value
  - of types
  - of location
  - of scoping (!)
- Know the difference between implicit and explicit declaration.
- Know what aliasing is.

What is a variable anyway?

- **Mathematical** variables represent a (possibly unknown) quantity or value. They usually are part of a model (or abstraction) of some concept or system. $f(x) = 2^{i\pi} - x$
- **Programming** variables are implementations of mathematical variables. (Has anyone here read Plato?)
  - Q: What is the difference between the mathematical variable i and a C++ variable `i`?
  - A programming variable has the following attributes:
    - Location
    - Type
    - Value
    - Scope

A variable's attributes can be assigned (or *bound*) to the variable at various times.

- Static binding — the attribute is determined at compile time.
  - Allows the compiler to "hard code" information about the variable into the executable code
  - Allows the compiler to perform optimizations based on its knowledge of the variable.
- Dynamic Binding — the attribute is determined at run time.
  - A variable's attribute could change during the course of execution, or remain undetermined—very flexible.
  - Information about the variable is usually stored with it.
  - Sometimes we *don't know* the value of the attribute at compile time.

Value

- The value attribute of a variable is most likely to be dynamic.
- Sometimes we want the value to be static. (Not to be confused with the `static` keyword in C.)

```
1 const int i = 2;
2
3 int foo(int j) { return i * j; }
4
5 int bar() {
6   int i = 10;
7   i = foo(i);
8   return i;
9 }
```


Identifiers vs. Variables

- This is the OCaml program is roughly equivalent to the previous.
- "Variables" in OCaml are *immutable* — there is no assignment statement.

```
1 let i = 2;;
2
3 let foo j = i * j;;
4
5 let bar () =
6   let i = 10 in
7   let i = foo (i) in
8       i;;
```


Static Typing

- Static Typing: the type of variables are known at compile time.
- This makes many operations very efficient.

```
1 int sqr(int i) {
2
3   return i * i;
4 }
```

```
1   movi r1, val(i)
2   movi r2, val(i)
3   multi r1,r2,r3
4   pushi r3
```

- The compiler can catch errors: improving programmer reliability.

```
1 string s = "hi";
2 bool b = true;
3 if s then printf("4") else printf("9");
```


Dynamic Typing

Some languages (e.g., BASIC, Perl, most shell, TCL) use dynamic typing.

```
1 #!/usr/bin/perl
2
3 $i = "The answer is ";
4 print "$i";
5
6 $i = 42;
7 print "$i\n";
```

Actually, Perl types are partially dynamic. Scalars, arrays, and hashes are represented with different syntax.

- Sometimes you want to be able to have both the advantages of strong typing *and* dynamic typing all at the same time.
- Methods include *overloading*, *templates*, and *automatic polymorphism*.

overloading

```
1 int identity(int i)  { return i; }
2 double identity(double x) { return x; }
```

templates

```
1 template <class T>
2 T ident(T &i)  return i;
```

automatic (ML)

```
1 # let id x = x;;
2 val id : 'a -> 'a = <fun>
```


- Heap allocated variables — completely dynamic
- Stack allocated variables — partially static "stack relative" allocation

```
1 int length() {
2   int i = 10;
3   String s = new String("hello");
4   return i + length(s);
5 }
```

- There is one language in which *all* variables — even function arguments — are allocated statically!

- Developed during a time with 4k was a lot of memory and processor speeds were measured in kHz.
- Looking up a memory location each time a variable is used is expensive!
- The problem: how do we get scientists to use a high level language rather than machine code?

Solution: Variable locations are hard-coded.

- This made Fortran almost as fast as assembly.
- Still the language of choice for numerical computation.
- Downside—you don't get recursion. (Modern Fortran fixes this.)

It is possible for multiple variables to refer to the *same* location.

```
1 int i = 20;
2
3 void inc(int &x) {
4   x = x + 1;
5 }
6
7 ... inc(i) ...  // i and x will be the same thing
```

Use with extreme caution!

Knowing about aliasing and storage is critical. *Never forget that your variables are representations only.*

```
1 int a[10];
2 int j;
3
4 void main () {
5   int i;
6   j = 42;
7   for(i = 0; i<=10; i++) a[i] = i;
8   printf("%d",j);
9 }
```

What will this print?

- Variables have a certain *scope* in the program for which they are valid.
- This allows us to have multiple variables with the same name.
- Usually the scope (or *lifetime*) is determined syntactically.

```
1 int foo(int i) {
2   int j = 10;
3   return j + 10;
4 }
5
6 int bar(int i) {
7   int j = 20;
8   return foo(j) + foo(i);
9 }
```


Consider the following program:

```
1 int i = 2;
2
3 int foo() { return i * i; }
4
5 int bar() {
6   int i = 10;
7   return foo();
8 }
```

- What value will function `bar` return?
  - 4
  - 100


```
1 (setq i 2)        ;; global variable i = 2
2
3 (defun foo ()
4   (* i i))
5
6 (defun bar ()
7   (let ((i 10))   ;; local variable i = 10
8     (foo)))       ;; call function foo
```

- What value will expression `(bar)` return?
  - 4
  - 100

Static vs. Dynamic Scoping

- Most languages use *static scoping*.
- Common LISP introduced dynamic scoping.
  - "It seemed like a good idea at the time."
  - It is considered to be a Bad Thing$^{\text{TM}}$ by most sentient life-forms.
- It's too easy to modify the behavior of a function.
- Correct use requires knowledge of a function's internals.

Still used by Lisp, some Scheme, and Emacs Lisp.

Problems

- (qdb 171) Which of the following is an advantage of dynamic typing that cannot be found with static typing?
  1. You don't have to declare types.
  2. No runtime type errors can occur.
  3. Dynamically typed code will run faster than statically typed code.
  4. None of these are advantages.
- (qdb 172) A C++ method can be either static or dynamic. How is this accomplished?

Answers

- Which of the following is an advantage of dynamic typing that cannot be found with static typing?
  1. You don't have to declare types.
  2. No runtime type errors can occur.
  3. Dynamically typed code will run faster than statically typed code.
  4. **Solution:** None of these are advantages.
- A C++ method can be either static or dynamic. How is this accomplished?
  Syntactically: A method is made dynamic via the `virtual` keyword. Implementation: the compiler uses a structure called a *vtable*.

Thanks

Proofreading help given by Eric Smith.